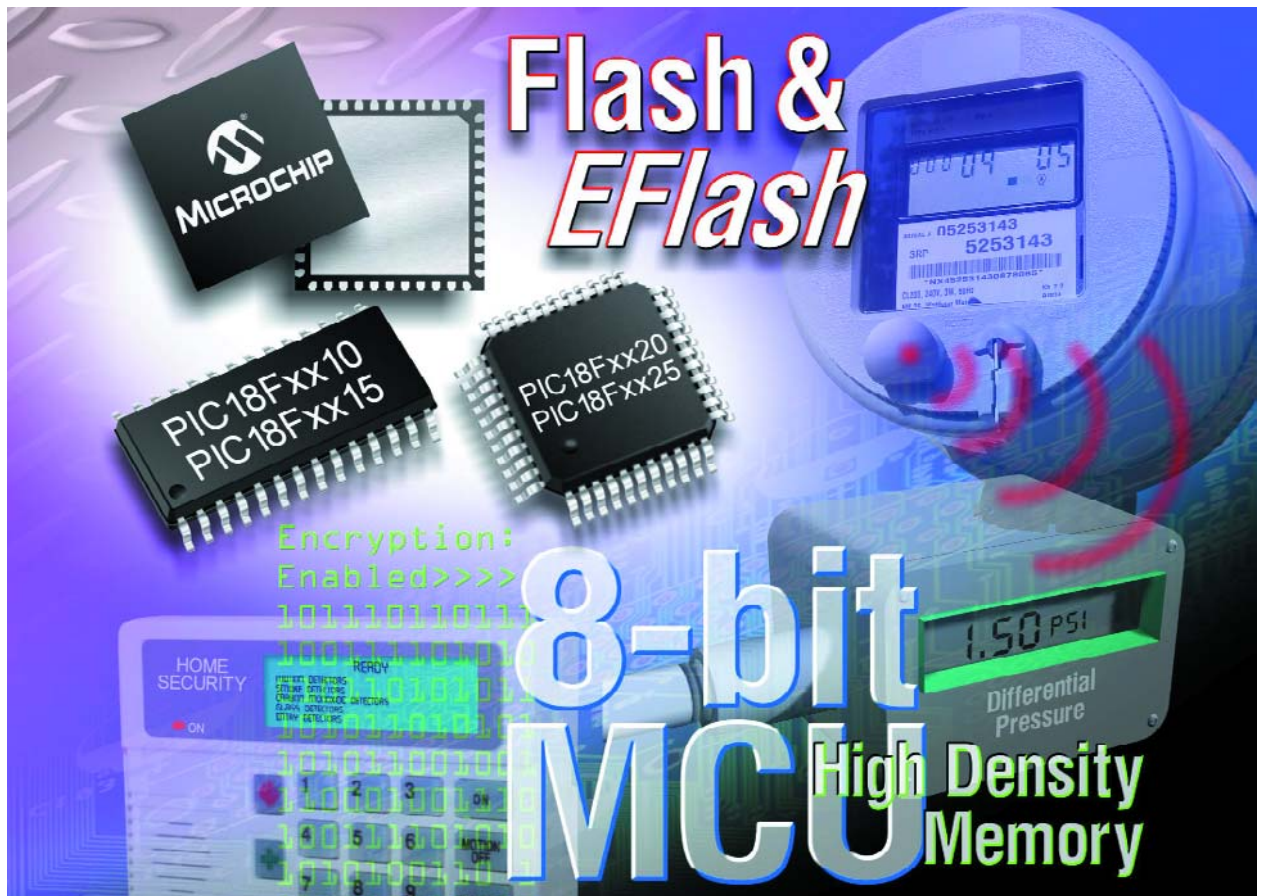


GOODBYE '16, WELCOME '17

Part 2: ROM (program memory) and long lookup tables

Ron Coates



The 18Fxx2 series of processors have either 16 k or 32 k of Flash ROM. However, each program instruction takes at least one word (i.e., two bytes) so in terms of the number of lines of program code you could write, you are looking at a maximum of 8 k or 16 k. Consequently, in terms of programming, you can only ever address even numbered ROM locations.

COME PIC18F

If you look at the program memory window for a 16F series program, the program address increments by one for each line, i.e., 0000, 0001, 0002 etc. The same code in an 18F would produce a program where each line incremented by two; 0000, 0002, 0004 etc. This doesn't matter in the normal course of events, but when we come to lookup tables, it does. The whole of the ROM memory is completely linear and there are no problems with CALL or GOTO instructions, no matter what the length of the program.

Look up tables

One of the features of PIC processors which has come in for some criticism in the past has been that lookup tables of any reasonable length were difficult to implement. One of the most useful features of the '18 series is that you can write lookup tables of virtually any length that will fit in the amount of Flash ROM available. It might be useful at this point to look in detail at the only way you could implement a look up table in the '16 series.

This was called a 'computed goto' by Microchip and required the programmer to prepare the lookup table as a subroutine starting with the command to add the value in the W register to the program counter. This was followed by a list of the values that were to be returned. To use the table you had to ensure that the value in W was the required table position before you called the subroutine. After the subroutine, the main program would have the looked up value in W.

The following example would return the maximum speed a driver should do in any intermediate gear. We will call the subroutine SPEED.

SPEED	ADDWF	PCL
	RETLW	0xFF
	RETLW	0x0A
	RETLW	0x20
	RETLW	0x3C
	RETLW	0x50

If the main program had the value of the gear (1, 2, 3 or 4) in W, then calling this subroutine would cause the program to jump to the line corresponding to the gear and return the maximum speed as a hex number. Note that the first line of the table corresponds to a value in W of zero. Since in this instance we would never have a value of zero, a dummy value (FF) is inserted.

The limitation of the above is that the value in W can only vary between 00 and FF (or 0 to 255 in decimal). And so you can only have 256 lines in the table. If you need a longer table than this, you have to split your table into 256 line chunks and use additional calculations to make sure you look up the right chunk. Clearly for values of several thousand, this is not practical.

The above method can still be used on the '18 series and is still useful for short tables. However the situation is even worse now, as the program counter only recognises even numbers. This means that whether W contains 0 or 1 makes no difference and the program will jump to the first line of the table (0xFF in the above example). For the program above to work properly with an 18F, you would need values of 2, 4, 6 and 8 in W. This also means that the maximum length of your table is now only 128.

If you need longer look up tables, you need to use an alternative method. This is described in detail in section 5 of the Data Sheet but these are general principles only. Any amount of the available Flash ROM can be used by the programmer as a lookup table (or series of lookup tables). So if we were using a chip with 32 k of ROM and we only needed 4 k for the program itself (remember this would be the equivalent of about two thousand lines of code), then 28 k would be available as look up table space. This is a very significant step forward and may well mean that applications which previously used an external EPROM to store data can now become true 'single-chip' applications with considerable savings in complexity, PCB size and, of course, cost.

There are also larger 18F chips available with up to 128 k of ROM. Consult the Microchip website for the latest information on these.

The ability to address this amount of space hinges on table pointers. There are three of these, TBLPTRU (Table Pointer Upper) only 5 bits of which are available, TBLPTRH (Table Pointer High) and TBLPTRL (Table Pointer Low). Together these are used to form a 21-bit long address which can access up to two megabytes. In practice, with the processors that we are talking about, TBLPTRU can be left at zero since the sixteen bits of the other pointers is more than sufficient to address the amount of ROM available.

These tables can be read and written to during program operation so you need to be careful with the setting of the pointers as it is perfectly possible to overwrite your program code.

So how do we get the data into these tables? This is a point where the data sheet is strangely silent. What you can't do is write it in with the program, in the way that you do with the computed go to. You can go to the Program Memory window in MPLAB and directly amend the values, and indeed this is useful for debugging purposes. However there is no way of saving these amendments and as soon as you exit the editor, the information is lost. In any case if you have got an eight kilobyte look up table you hardly want to input all the values by hand! We can think of two ways of doing it, and perhaps readers could suggest others. As an example let's take the values of a sound sampled musical instrument, for example, a piano. It would be normal to use a PC to manipulate and store data of this type, so we will assume that the table you want to get into the PIC is presently residing as a file in your PC.

If you can write a suitable program for the PC, then either the parallel or serial port could be used to download the file. The PICs we are discussing all incorporate a hardware USART (Universal Synchronous/Asynchronous Receiver Transmitter) which can easily be set up to communicate with the RS232 serial port of a PC.

If your skills do not extend to programming PCs, then an alternative method is to use an EPROM programmer to transfer the information to an EPROM, and then write a program in the PIC to transfer the information from the EPROM to the PIC's memory. This is rather long winded as it means you have to program in the code (using a PICSTART programmer or equivalent), then transfer the PIC to a board which has been made up just to transfer data from EPROM to PIC, before finally using the PIC in the target application.

Transferring an existing 16F application

Page one of the 18FXX2 data sheet has the rather optimistic statement that the 18F is 'source code compatible with 16C ...instruction sets'. This might make you think that all you have to do with an existing application is amend the references to the model number of the PIC and all will be well. In fact you will probably get a long list of error messages but it is not difficult to sort this out. First of all, the status register bits RPO and RP1 which were used for bank switching in the 16F, no longer exist. You will certainly have references to these as you could not otherwise have set up the I/O ports with the TRIS reg-

isters. All references to these must therefore be deleted. Another problem will exist only if your original program used indexed addressing. If it did, you will certainly have made references to the FSR registers. We now have three FSR registers labelled FSR0, FSR1 and FSR2 and each of these has a low and high byte. So all references to FSR will have to be changed to FSR0L. A similar problem will occur with references to the INDF register. These can be quickly sorted out by using the search and replace facility in the editor.

Finally, if you embed your CONFIG statements in the source code, these will also show up as an error. The CONFIG statements for the 18F series are completely different in format to the 16F and the best way is to delete the 16F statements and start again using the 18F INCLUDE file as a guide.

Syntax errors like these can be quickly sorted out. It is just as important that you review the program structure to make sure you are taking advantage of all the goodies these new chips provide.

Further outlook: DSP

Up until the beginning of this year, the 18F series were the most powerful processors that Microchip produced. That all changed a few months ago with the introduction of the dsPIC series. These are full 16 bit processors which also incorporate features usually found only on Digital Signal Processors (DSPs). As always *Elektor Electronics* will be the first to bring you details of these. Watch this space...

(040036-2)

See your design in print! *Elektor Electronics (Publishing)* are looking for *Freelance Technical Authors/Designers*

If you have

- * an innovative or otherwise original design you would like to see in print in Europe's largest magazine on practical electronics
- * above average skills in designing electronic circuits
- * experience in writing electronics-related software
- * basic skills in complementing your design with an explanatory text
- * a PC, email and Internet access for efficient communication with our in-house design staff

then do not hesitate to contact us for exciting opportunities in getting your designs published on a regular basis.

Elektor Electronics

K. Walraven, Head of Design Dept.

P.O. Box 75, NL-6190-AB Beek, The Netherlands, Fax: (+31) 46 4370161

Email: k.walraven@segment.nl

